OpenClaw Voice Integration

WhatsApp Sprachnachrichten mit OpenAI TTS

Technisches Handbuch

Version 1.0

Februar 2026

Inhaltsverzeichnis

1 Übersicht

Dieses System erweitert OpenClaw um automatische Sprachnachrichten-Generierung für WhatsApp. Wenn OpenClaw eine Textnachricht an einen konfigurierten Kontakt sendet, wird automatisch eine Sprachnachricht mit derselben Antwort generiert und zugestellt.

1.1 Anwendungsfall

Das System wurde für die **Seniorenberatung** entwickelt. Ältere Menschen bevorzugen oft Sprachnachrichten, da diese einfacher zu konsumieren sind als lange Textnachrichten. Die Stimme ist freundlich, spricht langsam und deutlich.

1.2 Funktionen

- Automatische Voice-Generierung mit OpenAI TTS (Text-to-Speech)
- Freundliche Frauenstimme (Nova) mit 0.9x Geschwindigkeit
- HD-Audioqualität für beste Verständlichkeit
- Automatischer Versand via WhatsApp (wacli)
- Multi-Kontakt-Unterstützung
- Autostart bei Mac-Login via LaunchAgent
- Deutsche Sprachausgabe garantiert

2 Architektur

2.1 Komponenten

Komponente	Beschreibung
`watcher.py`	Überwacht `gateway.log` auf neue Antworten
`responder.py`	Generiert MP3 via OpenAI TTS API
`sender.py`	Sendet MP3 via wacli an WhatsApp
`hook.py`	Definiert Voice-aktivierte Kontakte
LaunchAgent	Startet Watcher automatisch bei Login

Tabelle 1: Voice-System Komponenten

2.2 Datenfluss

1. WhatsApp-Nachricht kommt rein (via OpenClaw Gateway)
2. OpenClaw verarbeitet und antwortet mit Text
3. Gateway schreibt `Äuto-replied to +49..."` in `gateway.log`
4. Watcher erkennt neuen Log-Eintrag

5. Watcher liest letzte Assistant-Nachricht aus Session-Datei
6. Responder generiert MP3 via OpenAI TTS
7. Sender schickt MP3 via wacli an WhatsApp

2.3 Verzeichnisstruktur

```
~/.openclaw/
+-- skills/
|   +-- schadl-seniorenberatung/
|       +-- voice/
|           +-- __init__.py
|           +-- responder.py       # TTS-Generierung
|           +-- sender.py          # WhatsApp-Versand
|           +-- watcher.py         # Log-Ueberwachung
|           +-- hook.py            # Kontakt-Liste
+-- media/
|   +-- voice/                     # Generierte MP3s
+-- logs/
|   +-- gateway.log                # OpenClaw Gateway Log
|   +-- voice-watcher.log          # Watcher stdout
|   +-- voice-watcher.err.log      # Watcher stderr
+-- agents/
    +-- main/
        +-- sessions/              # Chat-Sessions (JSONL)

~/Library/LaunchAgents/
+-- com.openclaw.voice-watcher.plist  # Autostart
```

3 Voraussetzungen

3.1 Software

- macOS (getestet mit macOS 14+)
- Python 3.11+
- OpenClaw mit WhatsApp-Gateway
- wacli (WhatsApp CLI Tool)

3.2 API Keys

- OpenAI API Key mit TTS-Zugang

3.3 Python-Pakete

```
pip install openai
```

4 Installation

4.1 Verzeichnisstruktur anlegen

```
mkdir -p ~/.openclaw/skills/schadl-seniorenberatung/voice
mkdir -p ~/.openclaw/media/voice
mkdir -p ~/.openclaw/logs
```

4.2 OpenAI API Key konfigurieren

Option A: Umgebungsvariable (empfohlen)

```
# In ~/.zshrc einfuegen:
export OPENAI_API_KEY="sk-proj-..."
```

Option B: Credentials-Datei

```
# ~/.openclaw/credentials/openai.json
{
    "api_key": "sk-proj-..."
}
```

4.3 Voice-Dateien erstellen

Erstellen Sie die Python-Dateien im Verzeichnis:

`~/.openclaw/skills/schadl-seniorenberatung/voice/`

Die vollständigen Dateien finden Sie in Abschnitt 6.

5 Konfiguration

5.1 Kontakte hinzufügen

In `watcher.py` die `VOICE_CONTACTS` definieren:

```
# Voice-aktivierte Kontakte
VOICE_CONTACTS = {
    "+4916090895924": "Wolfgang Schadl",
    "+4915155561204": "Sandy",
}
```

5.2 TTS-Einstellungen

In `responder.py` können folgende Parameter angepasst werden:

Parameter	Wert	Beschreibung
VOICE	nova	Freundliche Frauenstimme
MODEL	tts-1-hd	HD-Qualität für beste Verständlichkeit
speed	0.9	Etwas langsamer für Senioren
LANGUAGE	de	Deutsch als Standardsprache

Tabelle 2: TTS-Konfigurationsparameter

5.3 Verfügbare Stimmen

Stimme	Charakteristik
alloy	Neutral
echo	Männlich
fable	Britisch
nova	Freundlich weiblich **(empfohlen)**
onyx	Tief männlich
shimmer	Sanft weiblich

Tabelle 3: OpenAI TTS Stimmen

5.4 LaunchAgent für Autostart

Erstellen Sie die Datei:

`~/Library/LaunchAgents/com.openclaw.voice-watcher.plist`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.openclaw.voice-watcher</string>
    <key>ProgramArguments</key>
    <array>
        <string>/Library/Frameworks/Python.framework/
            Versions/3.11/bin/python3</string>
        <string>/Users/mac/.openclaw/skills/
            schadl-seniorenberatung/voice/watcher.py</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>KeepAlive</key>
    <dict>
        <key>SuccessfulExit</key>
        <false/>
    </dict>
    <key>EnvironmentVariables</key>
    <dict>
        <key>PYTHONPATH</key>
        <string>/Users/mac/.openclaw/skills/
            schadl-seniorenberatung/voice</string>
```

```
        <key>OPENAI_API_KEY</key>
        <string>sk-proj-...</string>
        <key>HOME</key>
        <string>/Users/mac</string>
        <key>PATH</key>
        <string>/opt/homebrew/bin:/usr/bin:/bin</string>
    </dict>
    <key>StandardOutPath</key>
    <string>/Users/mac/.openclaw/logs/voice-watcher.log</string>
    <key>StandardErrorPath</key>
    <string>/Users/mac/.openclaw/logs/voice-watcher.err.log</string>
    <key>ThrottleInterval</key>
    <integer>10</integer>
</dict>
</plist>
```

5.4.1 LaunchAgent aktivieren

```
# Laden
launchctl bootstrap gui/$(id -u) \
  ~/Library/LaunchAgents/com.openclaw.voice-watcher.plist

# Entladen
launchctl bootout gui/$(id -u) \
  ~/Library/LaunchAgents/com.openclaw.voice-watcher.plist

# Status pruefen
launchctl print gui/$(id -u)/com.openclaw.voice-watcher
```

6 Dateien im Detail

6.1 responder.py – TTS-Generierung

Diese Datei ist das Herzstück der Voice-Generierung:

```
#!/usr/bin/env python3
"""Voice Responder fuer Seniorenberatung
OpenAI TTS mit freundlicher Frauenstimme nova
Optimiert fuer Senioren: langsames Tempo, klare Aussprache
"""

import os
import json
from pathlib import Path
from datetime import datetime
from openai import OpenAI


def get_openai_api_key() -> str:
    """Holt den OpenAI API Key aus verschiedenen Quellen."""
    # 1. Umgebungsvariable
```

```
    if os.getenv("OPENAI_API_KEY"):
        return os.getenv("OPENAI_API_KEY")

    # 2. OpenClaw Credentials
    cred_path = Path.home() / ".openclaw/credentials/openai.json"
    if cred_path.exists():
        with open(cred_path) as f:
            data = json.load(f)
            if "api_key" in data:
                return data["api_key"]

    raise ValueError("OPENAI_API_KEY nicht gefunden")


# Konfiguration
VOICE = "nova"          # Freundliche Frauenstimme
MODEL = "tts-1-hd"      # HD Qualitaet
LANGUAGE = "de"         # Deutsch
OUTPUT_DIR = Path.home() / ".openclaw/media/voice"


class VoiceResponder:
    """Generiert Sprachnachrichten fuer die Seniorenberatung."""

    def __init__(self, voice: str = VOICE, model: str = MODEL):
        self.client = OpenAI(api_key=get_openai_api_key())
        self.voice = voice
        self.model = model
        self.output_dir = OUTPUT_DIR
        self.output_dir.mkdir(parents=True, exist_ok=True)

    def generate(self, text: str, contact_id: str = "default") ->
       Path:
        """Generiert eine Sprachnachricht aus Text."""
        # Optimierungen
        text = self._ensure_german(text)
        text = self._optimize_for_seniors(text)

        # Generiere Audio
        response = self.client.audio.speech.create(
            model=self.model,
            voice=self.voice,
            input=text,
            speed=0.9  # Langsamer fuer Senioren
        )

        # Speichere Datei
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"seniorenberatung_{contact_id}_{timestamp}.mp3"
        filepath = self.output_dir / filename

        response.stream_to_file(str(filepath))
        return filepath

    def _ensure_german(self, text: str) -> str:
        """Stellt deutsche Aussprache sicher."""
        german_starters = (
            "hallo", "guten", "liebe", "sehr", "hier",
```

```
            "das", "die", "der", "ein", "eine", "ich",
            "wir", "es", "ja", "nein", "also", "nun"
        )

        first_word = text.strip().split()[0].lower() if
            text.strip() else ""

        if any(first_word.startswith(s) for s in german_starters):
            return text

        # Deutscher Praefix fuer Spracherkennung
        return f"Also: {text}"

    def _optimize_for_seniors(self, text: str) -> str:
        """Optimiert Text fuer bessere Verstaendlichkeit."""
        replacements = {
            "EUR": " Euro",
            "%": " Prozent",
            "z.B.": "zum Beispiel",
            "bzw.": "beziehungsweise",
            "usw.": "und so weiter",
            "d.h.": "das heisst",
            "u.a.": "unter anderem",
            "ca.": "circa",
            "inkl.": "inklusive",
            "MDK": "M D K",
        }
        for old, new in replacements.items():
            text = text.replace(old, new)
        return text


# Singleton
_responder = None

def get_responder() -> VoiceResponder:
    global _responder
    if _responder is None:
        _responder = VoiceResponder()
    return _responder


def generate_voice(text: str, phone: str = "default") -> str:
    """Hauptfunktion fuer OpenClaw Integration."""
    contact_id = phone.replace("+", "").replace(" ", "")[-6:]
    filepath = get_responder().generate(text, contact_id)
    return str(filepath)
```

6.2 sender.py – WhatsApp-Versand

```
#!/usr/bin/env python3
"""WhatsApp Voice Sender via wacli"""

import subprocess
import json
```

```python
import shutil
from pathlib import Path

# wacli Pfad - fuer LaunchAgent mit vollem Pfad
WACLI_PATH = shutil.which("wacli") or "/opt/homebrew/bin/wacli"


def send_voice_message(audio_path: str, to_phone: str) -> dict:
    """Sendet eine Sprachnachricht via wacli."""

    # Pruefe ob Datei existiert
    if not Path(audio_path).exists():
        return {
            "success": False,
            "error": f"Datei nicht gefunden: {audio_path}"
        }

    # Baue wacli Kommando
    cmd = [
        WACLI_PATH, "send", "file",
        "--to", to_phone,
        "--file", audio_path,
        "--mime", "audio/mpeg",
        "--json"
    ]

    try:
        result = subprocess.run(
            cmd,
            capture_output=True,
            text=True,
            timeout=60
        )

        if result.returncode == 0:
            try:
                output = json.loads(result.stdout) if
                    result.stdout else {}
            except json.JSONDecodeError:
                output = {"raw": result.stdout}

            return {
                "success": True,
                "to": to_phone,
                "file": audio_path,
                "output": output
            }
        else:
            return {
                "success": False,
                "error": result.stderr or result.stdout,
                "returncode": result.returncode
            }

    except subprocess.TimeoutExpired:
        return {"success": False, "error": "Timeout beim Senden"}
    except Exception as e:
        return {"success": False, "error": str(e)}
```

6.3 watcher.py – Log-Überwachung

Der Watcher ist die zentrale Komponente:

```
#!/usr/bin/env python3
"""Voice Response Watcher fuer Schadl Seniorenberatung"""

import os
import sys
import json
import time
import re
from pathlib import Path

# Unbuffered output fuer LaunchAgent
sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', buffering=1)
sys.stderr = os.fdopen(sys.stderr.fileno(), 'w', buffering=1)

# Voice-Module importieren
sys.path.insert(0, str(Path(__file__).parent))
from responder import generate_voice
from sender import send_voice_message

# Konfiguration
LOG_FILE = Path.home() / ".openclaw/logs/gateway.log"
SESSIONS_DIR = Path.home() / ".openclaw/agents/main/sessions"

# Voice-aktivierte Kontakte
VOICE_CONTACTS = {
    "+4916090895924": "Wolfgang Schadl",
    "+4915155561204": "Sandy",
}

CHECK_INTERVAL = 2
MIN_RESPONSE_LENGTH = 50
MAX_RESPONSE_LENGTH = 2000


def get_latest_session_file():
    """Findet die neueste Session-Datei."""
    if not SESSIONS_DIR.exists():
        return None
    sessions = list(SESSIONS_DIR.glob("*.jsonl"))
    if not sessions:
        return None
    return max(sessions, key=lambda p: p.stat().st_mtime)


def get_last_assistant_message(session_file):
    """Extrahiert die letzte Assistant-Nachricht."""
    try:
        with open(session_file) as f:
            lines = f.readlines()

        for line in reversed(lines):
            try:
                entry = json.loads(line.strip())
                if entry.get("type") == "message":
```

```
                    msg = entry.get("message", {})
                    if msg.get("role") == "assistant":
                        content = msg.get("content", "")
                        if isinstance(content, list):
                            for block in content:
                                if block.get("type") == "text":
                                    return block.get("text", "")
                        elif isinstance(content, str):
                            return content
            except:
                continue
        return None
    except Exception as e:
        print(f"[watcher] Fehler: {e}")
        return None


def clean_for_voice(text):
    """Entfernt Markdown und Formatierung."""
    text = re.sub(r'\*\*(.*?)\*\*', r'\1', text)
    text = re.sub(r'\*(.*?)\*', r'\1', text)
    text = re.sub(r'#{1,6}\s*', '', text)
    text = re.sub(r'\[([^\]]+)\]\([^\)]+\)', r'\1', text)
    return text.strip()


def process_response(phone, name):
    """Verarbeitet eine Antwort und sendet Voice."""
    session_file = get_latest_session_file()
    if not session_file:
        return False

    response_text = get_last_assistant_message(session_file)
    if not response_text:
        return False

    voice_text = clean_for_voice(response_text)

    if len(voice_text) < MIN_RESPONSE_LENGTH:
        return False

    if len(voice_text) > MAX_RESPONSE_LENGTH:
        voice_text = voice_text[:MAX_RESPONSE_LENGTH] + "..."

    print(f"[watcher] Generiere Voice...")
    voice_path = generate_voice(voice_text, phone)

    print(f"[watcher] Sende an {name}...")
    result = send_voice_message(voice_path, phone)

    if result.get("success"):
        print(f"[watcher] Erfolgreich!")
        return True
    else:
        print(f"[watcher] Fehler: {result.get('error')}")
        return False
```

```
def watch_log():
    """Hauptschleife: Ueberwacht Log auf neue Antworten."""
    print(f"[watcher] Starte fuer {len(VOICE_CONTACTS)} Kontakte")

    last_position = 0
    if LOG_FILE.exists():
        last_position = LOG_FILE.stat().st_size

    while True:
        try:
            if not LOG_FILE.exists():
                time.sleep(CHECK_INTERVAL)
                continue

            current_size = LOG_FILE.stat().st_size

            if current_size > last_position:
                with open(LOG_FILE) as f:
                    f.seek(last_position)
                    new_lines = f.readlines()
                    last_position = f.tell()

                for line in new_lines:
                    match = re.search(r'Auto-replied to (\+\d+)',
                        line)
                    if match:
                        phone = match.group(1)
                        if phone in VOICE_CONTACTS:
                            name = VOICE_CONTACTS[phone]
                            print(f"[watcher] Neue Antwort an
                                {name}")
                            process_response(phone, name)

            time.sleep(CHECK_INTERVAL)

        except Exception as e:
            print(f"[watcher] Fehler: {e}")
            time.sleep(CHECK_INTERVAL)


if __name__ == "__main__":
    watch_log()
```

7 Workflow-Beispiel

7.1 Sandy fragt nach Nachbarschaftshilfe

1. Eingehende Nachricht

Sandy schreibt via WhatsApp:

WhatsApp-Nachricht

„ich bin 75 jahre und kann kein auto mehr fahren, wohne in wabern, sag mir die nächste nachbarschaftshilfe“

2. OpenClaw antwortet

OpenClaw sendet eine Textantwort mit Kontaktdaten der Nachbarschaftshilfe.

3. Watcher erkennt

Im `gateway.log` erscheint:

```
2026-02-04T18:12:54Z [whatsapp] Auto-replied to +4915155561204
```

4. Voice wird generiert

Der Responder erstellt eine MP3-Datei:

```
~/.openclaw/media/voice/seniorenberatung_561204_20260204_181254.mp3
```

5. Voice wird gesendet

Sandy erhält zusätzlich zur Textnachricht eine Sprachnachricht.

8 Fehlerbehebung

8.1 Watcher läuft nicht

```
# Status pruefen
ps aux | grep watcher

# Logs pruefen
tail -50 ~/.openclaw/logs/voice-watcher.log
tail -50 ~/.openclaw/logs/voice-watcher.err.log

# Manuell starten zum Testen
python3 ~/.openclaw/skills/schadl-seniorenberatung/voice/watcher.py
```

8.2 wacli nicht gefunden

Im LaunchAgent muss der volle Pfad zu wacli verwendet werden:

```
# Pfad ermitteln
which wacli
# Typisch: /opt/homebrew/bin/wacli

# In sender.py pruefen:
WACLI_PATH = "/opt/homebrew/bin/wacli"
```

8.3 OpenAI API Fehler

```
# API Key testen
python3 -c "
from openai import OpenAI
client = OpenAI()
print(client.models.list())
"
```

8.4 Store is locked (wacli)

Wenn wacli blockiert ist:

```
# Alle wacli-Prozesse beenden
pkill -9 wacli

# Kurz warten, dann erneut versuchen
sleep 3
wacli send file --to ... --file ...
```

8.5 Voice-Datei manuell senden

```
# Letzte Voice-Datei finden
ls -lt ~/.openclaw/media/voice/ | head -5

# Manuell senden
wacli send file \
  --to 4915155561204 \
  --file ~/.openclaw/media/voice/seniorenberatung_561204_*.mp3
```

Tipp

Bei Problemen immer zuerst die Log-Dateien prüfen. Der Watcher schreibt alle Aktionen und Fehler in die Log-Dateien unter `~/.openclaw/logs/`.

9 Zusammenfassung

Datei	Funktion
`responder.py`	OpenAI TTS Generierung (nova, 0.9x, HD, deutsch)
`sender.py`	WhatsApp-Versand via wacli
`watcher.py`	Log-Überwachung, Multi-Kontakt-Support
`hook.py`	Voice-Kontakte Liste
LaunchAgent	Autostart bei Mac-Login

Tabelle 4: Übersicht aller Dateien

Ende des Handbuchs